April 2003

**Supervisor:**
Graham Kendall

**GP-GXK1 Group Members:**
Andrew Dodman (acd01u)
Peter Bloor (pab01u)
John Lord (jel01u)
Tze Yan Henry Leung (tyl01u)
Sikder Md Sazzadur Rahman (smr02u)

# "Win A Million!!!"
# Final Group Report
# GP-GXK1

**Abstract:**

This document is the final report for our Software Engineering Group Project, "Win a Million". The task involved attempting to achieve as high a score as possible from placing words of different scores onto a board. The report discusses all of the different elements of the project we have worked on, from the initial research to a detailed look at the complete program that actually generates a completed board. It also looks into the problems that we have faced along the duration of the project.

**Table of Contents and Appendix List:**

Abstract: 1
1. Project Brief: 3
2. The History of Scrabble: 3
3. A Detailed Brief of the "Win a Million" Game: 4
4. The Group Website: 4
5. The Publicly Accessible Website: 5
6. Game Strategies: 5
7. Data Structures: 6
7.1 Representation of the list of letter scores 6
7.2 Representation of the words 6
*Individual words* 6
*The lists of words* 7
*The amount of words of each word length* 7
7.3 The width, height and current best score 7
7.4 Representation of the tables 7
7.5 Data storage 7
8. The Core Program: 7
8.1 The reference class 7
8.2 File reading routines 9
*Letter scores* 9
*Word lists* 9
8.3 The solve method 10
*The findword method* 11
*The putword method* 11
*The wordfits method* 11
9. The Graphical User Interface: 12
9.1 The Previous Status of the Graphical User Interface 12
9.2 The Completed Graphical User Interface 12
9.3 Obtaining the List of Words and Letters 13
9.4 Error Checking with Input Information 13
9.5 Displaying the Completed Board 13
9.6 Dynamically Generating the Completed Board in HTML 14
10. Testing and Fine Tuning the Program 15
11. Project observations: 15
12. Conclusion: 16

Appendix A: Java Source Code for the Core Program
Appendix B: Java Source Code for the GUI
Appendix C: Screenshots of the GUI
Appendix D: Screenshots of the Group Website
Appendix E: Screenshots of the Public Flash Website
Appendix F: The User Manual
Appendix G: Minutes from our Formal Group Meetings

## 1. Project Brief:

The project involves a word game, similar to Scrabble, called "Win a Million" where the player must attempt to fit as many words from a given list onto a board, only using each word once. The words can be placed horizontally and vertically, and cross each other but there must be a blank space at the beginning and ending of the word. Each word is made up of a different combination of letters which each have a set value. It is these values that are calculated to give the score for the board, the aim being achieving the highest score possible. Our task as a group is to develop a program that is able to maximise the score of the puzzle in a set amount of time. The "Win a Million" word game problem that we are trying to solve is similar in many ways to the popular board game Scrabble. In order to better understand the problem we decided to investigate the history of Scrabble.

## 2. The History of Scrabble:

Scrabble was invented by an out-of-work architect from New York called Alfred Mosher Butts back in 1931. He attempted to create a game that would use both chance and skill, combining features of both anagram and crossword puzzles. He designed a board game, originally called "Lexico", and later renamed "Criss Cross Words". The game comprised of a board onto which a player would add tiles with letters on them to make up words. The idea was that the letters would have different values, and you were to try and create words from seven tiles you are given, aiming for a high score.
[*http://www.askoxford.com/wordgames/scrabble/?view=print*],
[*http://www.scrabble-assoc.com/info/history.html*].

As the game would involve different scores for letters based upon their usage, he needed a method for calculating values. This resulted in him scanning the New York Times to find frequencies of letters, therefore he could give the lower scores to the frequently appearing letters, and high scores to ones that appeared rarely. This method of analysis of our language and the original tile score allocation has remained valid for over half a century.

The game that we now know as Scrabble went into production when Alfred Butts met a games loving entrepreneur called James Brunot. The pair made some refinements on the rules and design and trademarked the game as *Scrabble ® Brand Crossword Game in 1948.* The actual name of the game means "to grope frantically."

The goal in Scrabble is to create words from a number of tiles, each marked with a letter, and worth different point values depending on the letter. These tiles are placed in a bag and each player chooses seven tiles randomly from it and attempts to make words. This is similar to the "Win a Million" puzzle in that they are both word board games and both involve letters having different values, the objective being to use as many high scoring letters as possible. In the puzzle we are using however, words are already given to the player and the

task is to place these different length words onto a grid. Also, "Win a Million" is a one-player game whereas Scrabble requires at least two players.

## 3. A Detailed Brief of the "Win a Million" Game:

"Win a Million", taken from the Contest Centre website [*http://www.contestcen.com/million.htm*] is a game similar to Scrabble and gives you a chance to win one million US dollars.

In the game, you are given an official word list containing 100 words in total, stored in 4 columns. Each column of the word list contains 25 words, each between 2 and 5 letters in length. Every letter in the word list has a specific value representing the score of each letter. Players must attempt to fit as many words from the official list as possible onto the board placing them horizontally or vertically, but they can only be used once each. The words are allowed to cross each other, as long as any new words created by doing this are present in the list. Unlike Scrabble, words do not have to touch each other and therefore it is possible to have unconnected words on different areas of the board. There must be a blank space, or the edge of the board, at the beginning or ending of each word, so that it is clear which words have been used.

The row score can be calculated by adding the letter values from each position in that row. For example: If the word on the first row is "AR" and the letter values of letter "A" and "R" are 3 and 5 respectively, then the row score will be 3 + 5 = 8. If there are no more words added to this row, then this number become the row score, and be added to the last column on the board. The total score for the board will be the sum of all the row scores.

The aim for the project is to create a program which will try and generate a good solution to the "Win a Million" game within a certain time limit whilst adhering to the "Win a Million" rules described above. We will be developing it using Java. Before attempting to solve the game, the program will need to load in a list of available letters and their scores, a list of words, the dimensions of the board and the time limit. Once the program has this information it will be ready to attempt to produce a good solution to the game. After the time limit has elapsed, a thread running alongside the program will interrupt it and return the best solution it has found. It will then display the completed board to the user showing this solution.

## 4. The Group Website:

We felt the group needed a way of displaying information and news about the Win a Million project, and decided as a group that the best way of doing this would be by having our own website [*SEE APPENDIX C*]. The site is relatively basic but it acts as a source for all the information that has been gathered over the term. After each formal meeting, the minutes are typed up by the secretary and are made available on the group site for viewing or download. This has been useful as we have been able to refer to the minutes when writing reports, so we know what stage we were at on the project at those dates.

The site holds a variety of information for download other than the minutes. Documents detailing the history of Scrabble are available and copies of Ashley Tuckers lecture notes are accessible too. When a member of the group finds a resource that is beneficial to the project the document or URL is passed on to John and the site is updated with a link to that information or the option for download, if it is a file rather than a web site.

A problem that did arise was that the site was initially hosted on the machines in the computer science building, and therefore it was only available locally. As most of the group live off campus this year, the site wasn't very helpful as we would have to go to the computer labs to check for information. It was obvious that we needed to host the site externally and the site is now hosted off campus. We can now access our group site from home, and this has helped all on the project to keep up to date with our progress.

## 5. The Publicly Accessible Website:

We decided that we should also have a website that was accessible to the public. This is a more graphical site than the group's site and has been created in flash by John. We felt that the site could be used on the open day, featured on a terminal next to the one running the program. The current design of this flash site will be similar to that of the GUI when it is completed. The banners used at the top of each will be the same so they look related. All buttons are animated so that the user is clear when they are interacting with them [*SEE APPENDIX D*].

The flash holds much of the general information of the project. It features the project brief, detailing the description of our task in programming the "Win a Million" puzzle. There is a section describing the rules of the game and a movie featuring the board shows how a board can be completed. A word list for September 2002's official puzzle is shown as an example of the words we may receive. Email addresses for everyone working on the project are available within the group information section. The site was developed at the beginning of the project as we were unsure whether we would have the time to create one nearer to the open day, as it has been a relatively time consuming process.

## 6. Game Strategies:

We have done a lot of research on the game Scrabble such as its history, the rules of the game, some source code of current Scrabble programs, and have also tried playing demos of these programs. The actual Scrabble game is not the same as the "Win a million" game but we found the programs were useful. The reason for playing the game demos is they helped us to design strategies and to get ideas for what algorithms and methods we should use in our program.

The basic strategy we came up with was first to put two words on a row, then have an empty row, then put two words on the next row, then have another empty row and repeat this all the way down the grid. The second part is to fit five letter and three letter words into any vertical space you can. This is explained more fully in 6.3, along with details of how it has been implemented.

## 7. Data Structures:

One of the fundamental decisions we had to make about our project was how we were going to represent the data we would be using. The data we need is:

- a list of letters of their scores
- a list of the words we can use
- the total number of each length of word
- the width and height of the board
- the current best score
- the current highest scoring table
- the table currently being worked on

### 7.1 Representation of the list of letter scores

The list of letters and their scores was a tricky one to work out. An original idea was to have a hash table which seemed the most logical way to store the data. The letter was a key and this mapped to an integer which contained the score. This idea was abandoned after implementing it proved surprisingly tricky. In the end, we reverted back to a very early idea of storing the letter scores in an array or integers, of size 26. Each letter maps to a certain index of the array, which is where the score for that letter will be stored. Access to the letter scores is done via functions, and so anyone coding routines which accessed this list would not have to know the implementation details.

### 7.2 Representation of the words

#### *Individual words*

We decided that words would be easiest to store as an object, so that information about that specific word could be self contained within a single object [*SEE APPENDIX A: The word class*]. This information is the word itself (a string), the score of the word (an integer) and a boolean which states whether or not that word has been put in the grid. Originally the word class contained a couple of functions relating to the state of the used variable. In the end these were removed and it was chosen to access the variable directly.

When the constructor for the object is called, it is passed a string and a pointer to the reference class. This string is the word to be stored. A for loop is then used to add together the scores for each letter to get the total score for the word.

***The lists of words***

The list of words is stored in a two dimensional array of Word objects. The first index in the array will specify the length of word to be used and the second index will specify a word from that list.

***The amount of words of each word length***

This is simply stored in an array of integers. Each index corresponds to a particular word length. The range of word lengths is four (2, 3, 4, 5) and so the array is declared with a size 4. To find the number of words of a certain length you just subtract two from the length. For example, to find out how many words of length two there are you access index 0.

### 7.3 The width, height and current best score

These are simply stored as integers.

### 7.4 Representation of the tables

The current highest scoring grid will be stored in a two dimensional array of characters as this is the most logical way to store it. This is because the representation of such an array on paper is a two by two grid. The current table being worked on is also stored in this way.

### 7.5 Data storage

These data objects clearly need to be available to be accessed from anywhere in the program. Therefore they will be stored in a reference class. The data can either be accessed by creating a new instance of this class (the variable are all static and so act as if there is only one global instance of each variable) or by passing a pointer to an instance of the class to another function.

## 8. The Core Program:

The program consists of three class files, MillionGui, Word and Ref. The main file which contains the GUI code and program code is called MillionGUI. Word is the object in which a word can be stored and Ref is the reference class where all the main global data is stored. The program runs from MillionGUI. John initially developed the Graphical User Interface to the program, and Andrew has tied in the routines to load in the word and letter lists, the code for the reference class and the method to provide solutions for the game.

### 8.1 The reference class

The reference class [*SEE APPENDIX A: the reference class*] contains the variables which are needed to be globally access by the program, and some functions. The variables it contains are as follows:

```
static int width, height, bestscore;
static int[] scores = new int[26];
static Word[][] wordlist = new Word[5][100];
static char[][] besttable;
static char[][] temptable;
static int totals[] = new int[4];
```

The width and height variables contain the width and height of the size of board which is to be produced. Bestscore contains the score of the highest table found so far. The scores array contains the letter scores. The wordlist array contains the lists of words. besttable is used to score the highest scoring table so far and temptable stores the table which is current being worked on. The totals array contains the total number of words of each length. They are all declared as static so that they have the same value throughout the program, even when called from different instances of the reference class.

Here is a description of the functions that the reference class contains:

```
public void setupBestTable(Ref refer, int x, int y)
```

This function is called just before the solve method starts. It sets the values of width and height and then initialises the besttable array to the correct size. This array is then filled with a '!' at every space. The '!' character denotes an empty space on the board.

```
public void setupTempTable(Ref refer, int x, int y)
```

This function does the same as above, except it works on the temptable array instead. It also does not set the values of height and width, it assumes they have already been set.

```
public int returnScore(char letter)
```

This function returns the score for a particular letter. If letter = '!' then it returns zero.

```
public void putScore(String letter, String score)
```

This function sets the score in the scores array for the letter in the string "letter" to the score in string "score". It takes a string as the argument for score as it is passed the raw data from the file. This is converted to an integer in the function using an Integer wrapper class. The array index is found by casting the string letter as an integer and subtracting 65.

```
public void setBestTable()
```

This function is used when the highest scoring table needs to be replaced. It simply copies the contents of temptable to besttable.

```
public void clearTempTable()
```

This function clears the contents of temptable by setting each location to '!' again.

```
public void jumble(int factor)
```

This function is used by the game solving algorithm. It slightly jumbles up each word list to increase the number of grids which can be produced. It swaps the first few elements (the number of elements is randomly selected) of each word list with a randomly chosen word from the list. It does this for words of every length. The factor argument specifies how many times this is done.

### 8.2 File reading routines

The bulk of these routines had already been written by John and it was Andrew's job to complete them so that they actually used the data which was read in. John coded them so that they read in each line of the file and printed it to the screen, which was an initial test to verify that the BufferedReader worked (discussed in more detail in 8.3). The letter scores must be read in first. This is because creating a word object needs to know these scores to calculate the score of the word.

#### *Letter scores*

This was a fairly simple routine to code (*SEE APPENDIX A: The letter file reading routine*). It first reads in a line from the file, and breaks from the loop if this is null. The code assumes that when the null is reached the letter list has been successfully read in. If this line isn't null then it reads in the next line. This is because the letter scores are stored such that the first line contains the letter and the proceeding line contains its score. It then passes these two strings to the reference class's putScore function to add them to the list.

#### *Word lists*

This is another fairly simple routine [*SEE APPENDIX A: The word file reading routine*]. It first reads in a line from the file. The end of the word lists file is marked by "*eof". This is used instead of null as we encountered problems when null was used. The index in the word list, where the word which has just been read in will be placed, is found by accessing the appropriate index of the totals array. This is found accessing the totals array at index (wordLine.length() – 2) with wordline being the string which has just been read in. The value in the totals array is then incremented as there is one more word in its list. The word can now be added into the correct list using the line:

```
newref2.wordlist[wordLine.length() - 2][pos] =
                                    new Word(wordLine, newref2);
```

### 8.3 The solve method

The solve method [*SEE APPENDIX A: the solve method*] that has been produced to create solutions consists of a few passes of the board and is supported by some functions. The method starts when the "Submit" button is pressed, and takes the width, height and available time from the GUI and puts them into variables. Underneath this we then started to code a method to produce solutions. The start of the code declares the variables needed, which are mainly integers, explained when we take a closer look at the code. An instance of the reference class is declared and so is a random number generator, seeded by the current system time.

The next stage of the solving code initialises the process. The time is multiplied by 1000 to get it into milliseconds, the besttable and temptable set up routines are called and the start system time is recorded. The process then starts and enters into a while loop. This loop terminates when the given number of milliseconds is reached, as determined by adding the amount of time available to the current system time.

The process of filling in the board starts with a kind of second initialisation step which resets everything for each iteration. It sets the current row being worked on to zero, clears temptable and sets all the words so that they have not been used. This is important so that errors are avoided when the board starts to be completed.

The next process is the horizontal pass. This runs a 'for' loop which places two words on alternate rows, starting with the first row. The length of the first word is selected to have a random length between two and four and the second word fills the remaining space in the row (meaning it can be up to five letters in length if there is enough space in the table). The findword method is then used to find each word and the putword method is used to place it into the temptable. The findword and putword methods are explained later in more detail.

The second pass is the first vertical pass. This places five letter words down the table in places where they can be arranged without violating the rules. It uses a 'for' loop to specify which row the process is on and works across the table in each row. An if statement firstly checks if the current position the process is on is a valid place to put a five letter word. If this is the case the findword algorithm is used to find if there is a word which fits in this position. If there is such a word it is placed onto the temptable.

The next pass is the second vertical pass. This has two alternate versions. Each version is effectively the same, except on how the last row is processed. On tables of an even height the last row will use two letter words and on tables of an odd height it will use three letter words. They are chosen between by finding the value of (v % 2), where v is the vertical dimension of the table. Again they first use if statements to find if the current position is a valid position to place a word, then find if a word exists which fits the position and places the word if it does. The if statements that determine if it is a valid position are more

complex on this pass as it has to take into account the five letter words which may have already been placed on the table.

The final pass is optional, and it only runs on a grid which is large enough. On wider tables, the whole width of the table will not be filled. This pass will fill in this space. It first works down the grid, starting from the left hand side of the empty space, and places words of different random lengths with spaces between them. It then finds, after checking if it is a valid position, any words which may be fitted in horizontally between these words.

The final part of the code generating the completed board checks if the score of the temptable is higher than the score of the besttable. If it is then it calls the setBestTable function and changes the value bestscore in the reference class. The final step jumbles up the word lists to increase the number of grids which may be generated.

After the process is finished and the program has managed to complete a board, the function createBoardArray is called which generates and displays the highest scoring table found.

***The findword method***

The findword method [*SEE APPENDIX A: The findword method*] is used to find a word which fits a given position in a table. It takes many arguments, which include the length of the word required, the start x and y coordinates of the word, the orientation of the word and the size of the table. It runs through a for loop and examines all the words of the given length. It checks if a certain word has been used and if it fits (using the wordfits function). If it does then it sets the words used value to true and returns its index in the list. If no word is found then it returns -1.

***The putword method***

This method [*SEE APPENDIX A: The putword method*] places a word into the temptable array. It takes similar arguments to the findword method, they are the length of the word, its position in the list, the orientation and its start coordinates. It uses two if statements which decide between the two placement methods (one for horizontal, one for vertical). The word is then placed into the table using a for loop. No error checking is needed as if the word did not fit the table properly it would not have been chosen by the findword method.

***The wordfits method***

This method *SEE APPENDIX A: The wordfits method*] determines whether or not a word fits in a certain position in the table. It is called from the findword method. It takes the same arguments as the findword method, except a certain position in the word list is also given to it. Again, two if statements chose between the horizontal and vertical variations of the method. It runs through each position a word will take in the table and compares it to the character at that position in the word. If they are the same in all positions (or some of the positions are empty spaces) then true is returned, otherwise false is returned.

## 9. The Graphical User Interface:

In the beginning, ideas for the layout of the Graphical User Interface were noted on paper using both ideas from group members and those already present in some computer Scrabble programs that were already available [*Networdz 1.0 from http://www.scrabble.4t.com/download.htm*]. In our program the Graphical User Interface is used to call the methods and functions within the core of the program which actually generate a completed board, so it acts as the controls for the system.

### 9.1 The Previous Status of the Graphical User Interface

The GUI has had major changes made from what we had developed by the end of the first Semester [*SEE APPENDIX C: Screenshots of the Graphical User Interface at the end of Semester One*]. Initially, we had only coded enough of the GUI so it had the ability to read in external files and extract necessary information from these files. It also had a basic layout for how the user would enter information that would be required for the running of the system. This involved several JTextFields where the height and width of the board could be entered. Back then, these fields were just there for aesthetics, and didn't actually do anything within the program.

### 9.2 The Completed Graphical User Interface

The look of the GUI has been modified so that it follows a similar style to the publicly accessible website that has been developed. Images located at the top and bottom of the user interface are also present within the flash site [*SEE APPENDIX C: Similarities between the GUI and the Flash Website*]. The look and feel of the program has been modified so that it no longer has a default Java appearance. Using a windows look and feel the interface looks much cleaner and has the advantage of displaying file type images within the JFileChooser, so you can tell by looking which program a file is associated with. [*SEE APPENDIX C: Screenshots of the completed Graphical User Interface*].

The main addition to the interface involves a large JEditorPane which is used for displaying information throughout the running of the program. It consists of a JEditorPane that displays HTML pages (discussed in more detail later) based on a URL that is provided. This allows the program to display pages of information containing the rules of the game and help information. A menubar has been added to the top of the GUI which has a file, edit and help menu. Other additions to the completed user interface include a log area located in the bottom right hand corner, which details the current processes of the system.

The overall structure of the completed GUI consists of 4 separate JPanels and a JMenuBar, all held within a JFrame, which together make up the whole of the visible interface. There are separate JPanels positioned at the top and the bottom of the main window,as well as one positioned to the left containing the 'fileChoicePanel' where the user inputs information about the board. The main JPanel holds the JEditorPane used to view HTML files.

### 9.3 Obtaining the List of Words and Letters

The external files are basically .txt files that, in the case of the list of words, are created so that there is one word per line. This makes it simpler to read in the files using the program. The user has to select two external files, one containing the full list of words and other containing the list of letters with their values. The GUI gives the user the option to browse through files using a JFileChooser, initially developed in the earlier part of the project. They can then select a file displayed within a window within the JFileChooser, and when the button "open" is clicked, the program reads in the file a line at a time using a BuffferedReader [*SEE APPENDIX B: readWordFile()*]. As the program reads in each file on a line by line basis, it is easy to separate the words, or letters, that are contained. When the program reads a word, a new word Object is created using the Word class.

The area of the GUI responsible for the reading in of files is called the 'fileChoicePanel' [*SEE APPENDIX B: createFileChoicePanel()*]. This is located to the left of the user interface and consists of five JTextfields, two 'browse' buttons that bring up the JFileChoosers discussed previously, and a 'submit' button. Two of the text boxes are greyed out as they are only used for displaying the names of the files that have been selected as containing the words and letter scores. The other three text fields allow the user to input the dimensions of the board, and the amount of time available to the program in milliseconds. The submit button extracts all of the information held within these JTextFields and performs several error checking processes to make sure that the data is valid.

### 9.4 Error Checking with Input Information

Once the submit button is pressed, the program checks to make sure that all of the five fields have been completed. A major part of the error checking looks at the board dimension information and the time that has been entered. It involves extracting the information from the JTextFields one by one, and trying to parse the information to an Integer. This is located within a try and catch statement. If the content of the JTextField cannot be converted into an Integer due to the content being of a different type or left blank, an exception is thrown and a JDialogBox Error message appears detailing the problem that has occurred.

### 9.5 Displaying the Completed Board

A major concern was how we would actually display the board of the game within the GUI once the program has completed its task. We were unsure whether it would be easier to make the completed board appear within the JFrame of the Graphical User Interface itself or have it appear in a pop up window. Another concern was how the GUI would display the board graphically, and what Java Swing components would be best used.

It was decided that we could enable the program to contain a window in which HTML could be displayed. This idea was influenced by a coursework task from

the previous year involving the creation of a simple web browser. John decided that a large area of the user interface could be set aside to display the completed board. This window involves the use of a JEditorPane that is able to display HTML and .txt files named '*webPanel*' that has a JScrollpane attached. The scrollpane has solved the problem of dealing with large boards as there are now vertical and horizontal scrollers that can be used to navigate a larger board.

As we are using an area capable of displaying simple HTML, we felt that we could show other information such as details about the program. We have developed some external HTML files that are displayed at different requests of the user. When the GUI initially loads up an HTML file is displayed featuring information about how to run the program. When the user selects the 'about' option from the 'help' menu, the content of '*webPanel*' is set to a new URL containing the 'about' HTML document.

### 9.6 Dynamically Generating the Completed Board in HTML

A board can be of any size, so it was important that we could dynamically create this as it was needed. So, we developed a system that could dynamically generate HTML files based on the input information provided, and the results that the program has found. This has been achieved by having a method which is activated once the program has generated a completed board and the Timer has finished its countdown.

The method writeHTML()[*SEE APPENDIX B: writeHTML()*] creates a file in the same directory as the GUI java file. It works by creating a FileOutputStream which creates the file itself. Initially there was just one filename, which was just rewritten over and over again each time the writeHTML() method was called. However, this caused problems as only the first generated board would be shown in the *'webPanel'*, despite the fact that the file being displayed contained the updated board. This is because Java loads the contents of the desired URL into memory. When the file is requested to be displayed in *'webPanel'* again, the display doesn't change, as the program retrieves the file from memory again, meaning we don't see the new content. This problem was relatively easy to resolve as it just meant creating a separate filename each time the method is called by adding a number to the end of the filename.

To actually create the HTML page the program needs to write bytes to this file using a DataOutputStream. Now we can just write strings to the file as plain HTML. This method basically creates a table in HTML and works by looping through the length of the array printing the information held to the external file. This is done row by row until the whole of the array has been processed, and the table within the HTML file is completed. As the completed board information is stored as an array, simple 'for' loops are used to progress through the array,printing out the information to the external file. Once completed, webPanel changes its content to URL of the new HTML page that has been created, displaying the completed board.

## 10. Testing and Fine Tuning the Program

Once we had created all of the bare structure of the program, we had to test it rigorously to make sure that everything was working as our design had planned it to. The code for the Graphical User Interface was woven in with the code that actually calculates and generates a completed board. Initial testing was only conducted on a board of 7 by 7, the size that the Contest Centre uses on its website. In the early stage of testing, we only had the program working for three letter words, as a random number generator had not yet been implemented to choose between words of different length.

After viewing the first stages of testing and seeing that the program was able to complete a board, it was time to expand the program, so that it would work for all words within the word list. The program was adapted so that it could now use words of lengths between 2 and 5 letters long, the lengths used in the Official word lists. One problem encountered was that the method responsible for calculating the total score for the board returned an incorrect answer. This was a simple error that had gone unnoticed for some time, but it was easily corrected.

We alternated the content of the external files, which contain the list of words, and the list of letters and their scores, in testing. This allowed us to make sure that we would be able to run the program without any difficulties on the open day, when this data is supplied to us. This involved changing the letters that words could be created from, and their scores, and the complete list of words. The program had no difficulty handling this extra data.

We were also having problems at certain times with the writeHTML() method. We edited the HTML several times to get a suitable look for the program and this led to problems on occasions, when the JEditorPane tried to display the completed board. If a whole column didn't have any letters in, the column would not be created in the HTML and the display seemed to show an illegal board. This problem was solved by adding a width attribute to the <td> tags when generating the HTML. Now, any empty cells will still have the same size as one which contains a letter.

## 11. Project observations:

We decided from the outset that as we did not know each other very well, we would wait to see if a natural leader emerged rather than appointing someone straight away. The early group meetings were brainstorming sessions with everyone contributing ideas as to the best way of solving the problem, which proved extremely effective. After about three weeks we had a rough idea of what we intended to do, and as a group we decided that it made sense to appoint the most experienced programmer as leader, Andrew. This decision was based around the fact that this person would best understand the program and would therefore be best placed to delegate tasks, especially those involving coding.

Very early on we exchanged telephone numbers and email addresses, and agreed that any new information or ideas we had would be relayed to the other members of the group one way or another. We also agreed to have two meetings per week to discuss and continue working on the project, one formal meeting with recorded minutes and one informal. A major breakthrough was the creation of the group website. This meant that all our useful information was stored in one place and as individuals we could check the site to find out what tasks were allocated and how the project was progressing. In this way we removed the problem of bad communication and the project moved forward more rapidly as a result.

The group meetings were sometimes poorly attended, for one reason or another. This was due in part to clashes with prior engagements and other commitments such as deadlines in other modules. Also, because we did decide on a plan early on, there was a general feeling in the group that the deadlines were quite far off, and there was no need for urgency or panic so it wouldn't matter if a few meetings were missed. However, this did not turn out to be a huge problem, largely because of good communication as a group outside of meetings, and having a website where the minutes of any missed meetings could be checked.

Another problem encountered during the project was slightly bad time management. We did not discuss targets early enough and didn't decide on deadlines for the major programming tasks until the second half of the project. This meant that in the early stages we did not have a structured approach as to what would be done and when it would be done by. We did however sort this problem out, and constructed a timeline detailing when we aimed to complete different parts of the project.

## 12. Conclusion:

Generally the group has worked well as a team. We have had some difficulties splitting the tasks between the five members, but this has changed as the project has drawn to a close. All members of the group have had a part to play in the presentation, open day and final report.

Overall, we are relatively pleased with the program we have produced. We would have preferred to have completed it before the Easter break so that we could have had extra time to try and implement more heuristics in an attempt to achieve higher scoring tables. We hope to have produced a program which will compete well against the rival groups program.

The project has given us a good insight into working as a team and what it may be like in a software development situation. It has made us appreciate the need for different team roles within a group and realise the difficulties that can arise. We all feel that it has been an extremely valuable learning experience.